INFORMATICĂ · PYTHON · CLASA A IX-A · PRELUCRĂRI ALE DATELOR NUMERICE · LECȚIA 7

Baze de numerație

Durată: 100 min Nivel: mate-info · Python Înainte: lecțiile py9-02

CE VEI ȘTI DUPĂ LECȚIE

- Explici ce înseamnă valoarea unei cifre în funcție de poziție și de bază.
- Transformi un număr din baza 10 în baza b (împărțiri repetate) și din baza b în baza 10 (puteri sau schema lui Horner), pe hârtie și în Python.
- Treci rapid între bazele 2 și 16 grupând câte 4 biți.
- Îți verifici rezultatele cu int(s, b), bin(), hex(), după ce ai scris singur algoritmul.

1. Ce este o bază de numerație

În baza 10, cifra are o valoare care depinde de **poziția** ei: $352 = 3\cdot10^2 + 5\cdot10^1 + 2\cdot10^0$. În baza `b` este la fel, doar că puterile sunt ale lui `b`, iar cifrele sunt `0, 1, …, b − 1`.

număr	calcul	în baza 10
$1011_{(2)}$	$1\cdot2^3 + 0\cdot2^2 + 1\cdot2^1 + 1\cdot2^0$	11
$213_{(4)}$	$2\cdot4^2 + 1\cdot4 + 3$	39
$2F_{(16)}$	$2\cdot16 + 15$	47

În baza 16 cifrele de la 10 la 15 se scriu `A, B, C, D, E, F`.

De ce ne interesează? Calculatorul memorează totul în baza 2 (biți). De exemplu, litera `A` are codul ASCII 65, iar în memorie apare ca `1000001`. Baza 16 este o prescurtare comodă pentru baza 2: culorile din pagini web (`#FF8800`), adresele de memorie și adresele MAC se scriu în hexazecimal.

2. Din baza 10 în baza b: împărțiri repetate

Împarți numărul la `b` cât timp nu a ajuns la 0 și reții resturile. **Resturile, citite de la ultimul la primul**, sunt cifrele în baza `b`. Pentru `13` în baza 2:

`n`	`n // 2`	`n % 2`
13	6	**1**
6	3	**0**
3	1	**1**
1	0	**1**

Citit de jos în sus: $13 = 1101_{(2)}$. De ce invers? Primul rest este ultima cifră (cea a unităților), exact ca la `n % 10` în baza 10.

Varianta 1: rezultatul ca număr (pentru $b \le 10$). Fiecare rest se pune **în fața** celor de până acum, cu ajutorul puterii lui 10, `p`: este tehnica „adaug o cifră la stânga” pe care o știi de la prelucrarea cifrelor.

```python
n, b = 13, 2
r, p = 0, 1
while n > 0:
    r += (n % b) * p      # cifra nouă merge în stânga
    p *= 10
    n //= b
print(r)                  # 1101
```

Varianta 2: rezultatul ca text (merge și pentru `b = 16`). Cifra o luăm dintr-un șir de caractere, după poziție:

```python
CIFRE = "0123456789ABCDEF"
n, b = 2026, 16
s = ""
while n > 0:
    s = CIFRE[n % b] + s  # lipim cifra în stânga
    n //= b
if s == "":
    s = "0"               # n = 0 nu intră deloc în buclă
print(s)                  # 7EA
```

ATENȚIE — GREȘELI FRECVENTE

Pentru `n = 0` bucla nu se execută și ai obține `""` (sau 0 la varianta 1, ceea ce e corect din întâmplare). Testează mereu cazul `n = 0`.

3. Din baza b în baza 10

Metoda puterilor (de la dreapta la stânga): fiecare cifră se înmulțește cu puterea corespunzătoare a lui `b`. Dacă numărul în baza 2 este dat ca întreg, de exemplu `1101`, îi iei cifrele cu `% 10` și `// 10`:

```python
n, b = 1101, 2
v, p = 0, 1
while n > 0:
    v += (n % 10) * p     # ultima cifră · b la puterea poziției
    p *= b
    n //= 10
print(v)                  # 13
```

Schema lui Horner (de la stânga la dreapta): pornești cu `v = 0` și, pentru fiecare cifră, faci `v = v · b + cifra`. Merge foarte bine când numărul este dat ca text.

```python
CIFRE = "0123456789ABCDEF"
s, b = "7EA", 16
v = 0
for c in s:
    v = v * b + CIFRE.index(c)   # index() dă poziția caracterului: "E" → 14
print(v)                          # 2026
```

cifra citită	`v` după pas
`7`	`0·16 + 7 = 7`
`E`	`7·16 + 14 = 126`
`A`	`126·16 + 10 = 2026`

SFAT

Horner face o singură înmulțire pe cifră și nu are nevoie de puteri. Este aceeași idee ca la construirea unui număr din cifre în baza 10: `v = v * 10 + c`.

4. Baza 2 ⇄ baza 16, fără baza 10

Pentru că $16 = 2^4$, fiecare cifră hexazecimală corespunde exact unui grup de **4 biți**.

hexa	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
biți	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Exemplu. $11111101010_{(2)}$: grupezi de la **dreapta** câte 4: `111 | 1110 | 1010` → `7 | E | A`, deci $7EA_{(16)}$. Invers, $2F_{(16)} = 0010\ 1111_{(2)} = 101111_{(2)}$.

La fel, baza 8 grupează câte **3** biți ($8 = 2^3$).

5. Ce oferă Python (doar pentru verificare)

După ce știi algoritmul, poți verifica rapid un rezultat:

apel	rezultat	ce face
`bin(13)`	`'0b1101'`	text în baza 2, cu prefixul `0b`
`hex(2026)`	`'0x7ea'`	text în baza 16, cu prefixul `0x`
`oct(64)`	`'0o100'`	text în baza 8
`format(47, "X")`	`'2F'`	fără prefix, cu majuscule (`"b"` pentru baza 2)
`int("1101", 2)`	`13`	text în baza b → număr
`int("2f", 16)`	`47`	acceptă și litere mici
`int("12", 2)`	`ValueError`	`2` nu este cifră în baza 2

ATENȚIE — GREȘELI FRECVENTE

`bin()` și `hex()` întorc **text**, nu numere: `bin(5) + 1` dă eroare. Iar `int("1101")`, fără bază, este o mie o sută unu, nu 13. Prefixul `0b` îl tai cu o *felie*: `s[2:]` este textul `s` fără primele două caractere (`bin(13)[2:]` este `'1101'`); felierea o studiem la liste, în lecția py9-15.

LA BAC

La Bac (și la evaluările cu probleme de tip pbinfo) se cere algoritmul scris de tine. Funcțiile de mai sus sunt utile ca să-ți verifici testele, nu ca soluție.

6. Testarea unui program de conversie

caz	exemplu (10 → 2)	de ce
obișnuit	`13 → 1101`	
zero	`0 → 0`	bucla nu se execută
unu	`1 → 1`	o singură cifră

caz	exemplu (10 → 2)	de ce
putere a bazei	`64 → 1000000`	multe zerouri la final
cu o unitate mai puțin	`63 → 111111`	numai cifre maxime
invers (verificare)	`int(rezultat, 2) == n`	„dus-întors” pe sute de valori

Ultimul rând este un test foarte puternic: transformi `n` în baza `b` cu programul tău, apoi înapoi cu `int(…, b)`, pentru toate valorile `n` de la 0 la 1000. Dacă obții mereu `n`, programul este aproape sigur corect.

7. Verifică-te

1. Transformă `45` în baza 2 și în baza 16, pe hârtie.
2. Cât este $1100100_{(2)}$ în baza 10?
3. Ce cifre poate avea un număr scris în baza 7? Care este cel mai mare număr de 3 cifre în baza 7, scris în baza 10?
4. Scrie $A5_{(16)}$ în baza 2, fără să treci prin baza 10.
5. Ce afișează `print(int("101", 2) + int("101"), bin(6)[2:])`?
6. Câte cifre are numărul `1000` scris în baza 2?

Răspunsuri

1. $45 = 101101_{(2)} = 2D_{(16)}$.
2. $64 + 32 + 4 = 100$.
3. Cifrele 0…6. $666_{(7)} = 7^3 - 1 = 342$.
4. $A = 1010$, $5 = 0101$, deci $10100101_{(2)}$.
5. `106 110`: $5 + 101 = 106$, iar `bin(6)` este `'0b110'`, din care `[2:]` taie primele două caractere.
6. 10 cifre: $2^9 = 512 \le 1000 < 1024 = 2^{10}$ ($1000 = 1111101000_{(2)}$).

Probleme de lucru în clasă (pbinfo)

Le rezolvăm împreună în ședință, în ordinea de mai jos.

- [ ] **#429** **cifbin** — ușoară
  https://www.pbinfo.ro/probleme/429/cifbin
- [ ] **#428** **transfb** — ușoară
  https://www.pbinfo.ro/probleme/428/transfb
- [ ] **#426** **bazab** — ușoară
  https://www.pbinfo.ro/probleme/426/bazab

Temă pentru acasă (pbinfo)

Trimite soluțiile pe pbinfo și notează punctajul obținut. Bifează ce ai terminat.

- [ ] **#945** **baze** — ușoară
  https://www.pbinfo.ro/probleme/945/baze
- [ ] **#947** **baza16** — ușoară
  https://www.pbinfo.ro/probleme/947/baza16

- [ ] #122 **cifrebinare** — ușoară
  https://www.pbinfo.ro/probleme/122/cifrebinare
